Introduction to R

Selahattin Murat Sirin

08/09/2020

Outline

- R Ecosystem
- R Basics
- Data Analysis
- Graphics
- R Markdown

BASIC R OPERATIONS

R Ecosystem

R is a free statistical computing software. It can be downloaded from CRAN website for free (https://cran.r-project.org/). The latest version is R 4.0.2. If you have difficulties in installing the latest version, you can install older versions from (https://cran.r-project.org/src/base/R-3/) - the last is the 3.6.3.

RStudio's IDE (Integrated Development Environment) offers a very versatile user interface. There are other IDEs; however, RStudio is the most popular among R users. It can be installed from (https://rstudio.com/)

Most people want to use cloud services to save their work. RStudio can interface with Git(Hub). You can create a free GitHub account from (https://github.com/), and save your work in there.

Functions and Packages

R is an object-oriented programming which organizes software design around data, or objects, rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

Each function has 6 elements:

1) Name and description
2) Usage
3) Arguments
4) Details
5) Values
6) Dependencies and references

```
## Check the sum function
sum
```

```
?sum
```

All functions are provided through packages, and it provides significant flexibility. On the other hand, unlike other statistical

Example

An example using `install.packages` function

```r
install.packages("dplyr", dependencies = TRUE)
```

Load installed package using `library` or `require` functions

```r
library(dplyr)
require(dplyr)
```

You can also load package useing `p_load` function provided in **pacman** package. When sharing your work with others, this might be a better solution.

```r
install.packages("pacman")
library(pacman)
p_load(dplyr)
```

Use `update.packages` function to update packages.

Session Info

Always check your session before starting any project

```r
## Provides information about session
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_Canada.1252  LC_CTYPE=English_Can
## [3] LC_MONETARY=English_Canada.1252 LC_NUMERIC=C
## [5] LC_TIME=English_Canada.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  me
##
```

R objects

```
a <- 5; b <- "a"; c <- TRUE
A <- c(1:5)
B <- c("a", "b", "c")
```

```
class(a); class(b); class(c)
```

```
## [1] "numeric"

## [1] "character"

## [1] "logical"
```

```
class(A); class(B)
```

```
## [1] "integer"

## [1] "character"
```

Dates

```
date1 <- as.Date("10/23/2016", format = "%m/%d/%Y")
date2 <- as.Date("23 October 16", format = "%d %B %y")
date3 <- as.Date("11/10/2016", format = "%m/%d/%Y")
```

```
date1 - date2
```

```
## Time difference of 0 days
```

```
weekdays(date2)
```

```
## [1] "Sunday"
```

```
as.numeric(date3)
```

```
## [1] 17115
```

Besides base operations, **lubridate** package allows multiple operations with dates.

Mathematical operations

```
5 + 5 # Addition
5 - 5 #  Subtraction
5 * 5 # Multiplication
5 / 5 # Division
0 / 0 # Division: Not a number (NaN)
pi # Pi
sqrt(5) # Square root
5^2 # Exponent
5**2 # Exponent
5 %/% 2 # Division
5%%2 # Modulo
abs(-5) # Absolute value
log(5) # ln
exp(5) # Exponent
log10(5) # log10
log2(5) # log2
round(5/2, digits = 2) # Rounding
signif(12345.6789, digits = 5) # Rounding
ceiling(5/2) # The higher integer
```

Mathematical operations

Lag and Lead

```
library(dplyr)
x <- seq(1:10)
lag(x,1)
```

```
##  [1] NA  1  2  3  4  5  6  7  8  9
```

```
lag(x,2)
```

```
##  [1] NA NA  1  2  3  4  5  6  7  8
```

```
lead(x,1)
```

```
##  [1]  2  3  4  5  6  7  8  9 10 NA
```

```
lead(x,2)
```

```
##  [1]  3  4  5  6  7  8  9 10 NA NA
```

Mathematical operations

Cumulative Sum, Cumulative Product, Extremes

```
cumsum(x)
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55
```

```
cumprod(x)
```

```
##  [1]       1       2       6      24     120     720
## [10] 3628800
```

```
cummax(x)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
cummin(x)
```

```
##  [1] 1 1 1 1 1 1 1 1 1 1
```

```
cummean(x)
```

```
##  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5
```

Mathematical operations

```r
x <- 1:20
first(x)
```

```
## [1] 1
```

```r
last(x)
```

```
## [1] 20
```

```r
nth(x,9)
```

```
## [1] 9
```

```r
range(x)
```

```
## [1]  1 20
```

R Operators

- <- Assign value
- = Assign value
- : Generate regular sequences
- c() Combine Values into a Vector or List
- (<) Smaller
- (>) Greater
- <= Smaller or equal
- = *Greater or equal*
- == Equal
- != Not equal
- Or
- & And

Matrix operations

- `matrix()` Creates matrix
- `rbind()` Bind rows
- `cbind()` Bind columns
- `ncol()` Number of columns
- `nrow()` Number of rows
- `t()` Transpose of a matrix
- `det()` Determinant of a matrix
- `eigen()` Eigen values
- `solve()` Solves equation
- `diag()` Construct a diagonal matrix
- `A%*%B` Dot product
- `A%o%B` Outer product
- `crossprod(A,B)` Return a matrix cross-product
- `kronecker()` Computes the generalised kronecker product

Matrix examples

```r
A <- matrix(c(2,5,7,1,6,12,3,2,2), byrow = TRUE, ncol = 3)
a <- c(2,5,7)
b <- c(1,6,12)
c <- c(3,2,2)
B <- rbind(a,b,c)
t(A)
solve(A)
eigen(A)
diag(4)
```

To select an element in a matrix you can use `A[2,3]`. This gives the element in the second row and third column

Data structure

- Vector (Atomic) is the fundamental data structure. There are six basic vector types: (1) Logical, (2) Integer, (3) Double, (4) Character, (5) Complex, (6) Raw. Use `c()` or `vector()` functions to create vectors. The elements of a vector are all of the same type.
- List is referred as generic vector. The elements may be different types. Use `list()` to create a list.
- Matrix is a two dimensional array. Use `matrix()` to create a matrix.
- Array is a multi-dimensional vector. All elements should be the same type. Use `array()` to create an array

Data frame

- Data frame is stored as a list of vectors, factors and/or matrices each of which has the same length. Use `data.frame()` to create a data frame.

```
df <-  data.frame(a = c(1:10), b = rnorm(10, 0, 0.5), c = r
```

Table 1: Data Frame

a	b	c
1	-0.6170387	1
2	-0.2865482	2
3	0.4057937	1
4	0.6413598	2
5	0.1358248	1
6	0.0568945	2

Basic data functions

- `head()` returns the first part of a vector
- `tail()` returns the first part of a vector
- `names()`,`colnames()` displays column names
- `dim()` retrieves the dimension of an object
- `str()` displays the structure of an R Object
- `table()` used for cross tabulation
- `xtabs()` creates a contingency table
- `margin.table()` creates contingency table in array form
- `prop.table()` express table nntries as a fraction of marginal table
- `ftable()` creates ‘flat’ contingency tables

Basic data functions - Examples

```r
library(datasets)
data(mtcars)
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1
```

Basic data functions - Examples

In basic R operations, you can use `with()` function to evaluate an expression in an environment constructed from data.

```
## Create cross-table
with(mtcars, table(cyl, gear))
```

```
##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

Missing data

- `complete.cases()` returns a logical vector indicating which cases are complete
- `na.omit removes()` removes rows with missing data

```
missing.values <-  c(1,2,NA, 4, 5, NA, 7, NA, NA)
complete.cases(missing.values)
```

```
## [1]  TRUE  TRUE FALSE  TRUE  TRUE FALSE  TRUE FALSE FALS
```

```
na.omit(missing.values)
```

```
## [1] 1 2 4 5 7
## attr(,"na.action")
## [1] 3 6 8 9
## attr(,"class")
## [1] "omit"
```

Many packages provide functions for handling missing data. Check **dplyr, Amelia, Hmisc, missForest** packages for examples.

Import Data

- `read.table(file)`
- `read.csv(file)`
- `read.csv2(file)`
- `read.delim(file)`
- `read.delim2(file)`

Check **readr, XLConnect** packages to import data from excel files

Import Data

You can import data from other statistical programs using **foreign** and **Hmisc** packages

```
library(foreign)
library(Hmisc)
# SPSS
data.spss <- read.spss("File name")

# SAS
data.sas <- sasxport.get("File name")

# Stata
data.stata <- read.stata("File name")

# Systat
data.systat <- read.systat("File name")
```

BASIC DATA OPERATIONS

Rename variable

Renaming a variable using `names()` function.

```r
library(datasets)
data(mtcars)
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "v
## [11] "carb"
```

```r
names(mtcars)[2] = "lyc"
names(mtcars)
```

```
##  [1] "mpg"  "lyc"  "disp" "hp"   "drat" "wt"   "qsec" "v
## [11] "carb"
```

Rename variables

Tidyverse is a collection of packages that are essential for data analysis. One of the most useful packages is the **dplyr** package. `rename()` package helps to rename variables.

%>% function is the pipeline function called from **magrittr** package. It simplifies analysis to a great extent.

```r
library(datasets)
library(dplyr)
data(mtcars)
mtcars <- mtcars %>%
  rename(MilesPGallon = mpg)
names(mtcars)
```

```
##  [1] "MilesPGallon" "cyl"          "disp"         "hp"
##  [6] "wt"           "qsec"         "vs"           "am"
## [11] "carb"
```

Create a new variable

The base R function has $ operator that acts on vectors, matrices, arrays and lists to extract or replace parts.

```
library(datasets)
data(mtcars)
mtcars$mpgc <- mtcars$mpg / mtcars$cyl
summary(mtcars$mpgc)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.300   1.928   3.108   3.837   5.700   8.475
```

We can do the same with `mutate()` function in **dplyr** package.

```
library(dplyr)
library(datasets)
data(mtcars)
mtcars <- mtcars %>%
  mutate( mpgc = mpg /cyl)
```

Ordering variables

The base `order()` function rearranges the data set. `attach()` function attaches the dataset to the environment, so objects in the database can be accessed by simply giving their names. **dplyr** package has `arrange()` function for the same purpose, and it is more convenient.

```
library(datasets)
data(mtcars)
attach(mtcars)
mtcars[order(mpg, decreasing = TRUE),] %>%
  head()
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am
## Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1
## Fiat 128       32.4   4  78.7  66 4.08 2.200 19.47  1  1
## Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1
## Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1
```

Merging datasets

The base merge() function merges data sets by common columns or row names. Similarly, _join() functions in dplyr packages offer more options. Compare these two data sets using merge() function.

```
## Create two data frames
df_1 <- data.frame(id = c(1,2,3), value = rnorm(3,1,0.2))
df_2 <- data.frame(id = c(3,4,5,6), value = rnorm(4,1,0.2))
new_df <- merge(df_1, df_2, by = "id")
new_df_1 <- merge(df_1, df_2, by = "id", all = TRUE)
print(new_df)
```

```
##   id   value.x  value.y
## 1  3 0.8031866 1.218372
```

Merging datasets using join functions

```
inner_join(x, y, by = NULL)
left_join(x, y, by = NULL)
right_join(x, y, by = NULL)
full_join(x, y, by = NULL)
anti_join(x, y, by = NULL)
```

```
library(dplyr)
inner_join(df_1,df_2,by="id")
```

```
##   id   value.x  value.y
## 1  3 0.8031866 1.218372
```

Subsetting data

- `X[n]` select nth element; nth row if dim > 2
- `X[n,k]` select element from nth row, kth column
- `X[-n]` All but nth element
- `X[1:n]` First n elements
- `X[c(a,b,c)]` select multiple elements
- `X[x > k]` selects elements greater than k
- `X[x > k | x < k]` selects elements greater than or less

```r
library(datasets)
data(mtcars)
mtcars$mpg[10:15]
```

```
## [1] 19.2 17.8 16.4 17.3 15.2 10.4
```

```r
mtcars$mpg[mpg > 30]
```

```
## [1] 32.4 30.4 33.9 30.4
```

Subsetting data

The base `subset()` function and `filter()` or `select()` functions from **dplyr** package offers more flexibility when subsetting data.

```
library(datasets)
data(airquality)
subset(airquality, Temp > 80, select = c(Ozone, Temp)) %>%
  head()
```

```
##    Ozone Temp
## 29    45   81
## 35    NA   84
## 36    NA   85
## 38    29   82
## 39    NA   87
## 40    71   90
```

Summarizing data sets

Use summary() function to summarize variables

```
data(mtcars)
summary(mtcars[,1:4])
```

```
##       mpg             cyl             disp             h
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu
##  Median :19.20   Median :6.000   Median :196.3   Median
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.
```

Summarizing data sets

Row and column sums and menas can be retrieved by these functions:

```
colSums (x, na.rm = FALSE, dims = 1)
rowSums (x, na.rm = FALSE, dims = 1)
colMeans(x, na.rm = FALSE, dims = 1)
rowMeans(x, na.rm = FALSE, dims = 1)
```

```
library(datasets)
data(mtcars)
colMeans(mtcars)
```

```
##        mpg        cyl       disp         hp       drat
##  20.090625   6.187500 230.721875 146.687500   3.596563
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

Note: There are many packages that offer summary tables such as **stargazer**.

Stargazer package

```r
library(stargazer)
library(dplyr)
library(datasets)
data(mtcars)
mtcars %>%
  select(hp, mpg,disp) %>%
  stargazer(., median = TRUE,
            iqr = TRUE, type = "text")
```

```
##
## =======================================================
## Statistic N   Mean   St. Dev. Min Pctl(25) Median Pctl(7
## -------------------------------------------------------
## hp        32 146.688  68.563  52    96.5    123     180
## mpg       32 20.091   6.027   10    15.4    19.2    22.8
## disp      32 230.722 123.939  71   120.8   196.3    326
## -------------------------------------------------------
```

Reshaping data sets

To reshape data, **tidyr** package has `pivot_()` functions (pivot_longer and pivot_wider). Assume we have 3 companies with daily average stock prices in USD.

```
require(tidyr)
set.seed(23)
stocks <- data.frame(
  DATE = as.Date('2017-01-01') + 0:9,
  A = as.integer(rnorm(10, 5, 1)),
  B = as.integer(rnorm(10, 10, 1)),
  C = as.integer(rnorm(10, 15, 1)))
stocks.long <- pivot_longer(stocks, A:C, names_to = "Compar
                            values_to = "Price")
head(stocks.long)
```

```
## # A tibble: 6 x 3
##   DATE       Companies Price
##   <date>     <chr>     <int>
## 1 2017-01-01 A             5
```

Strings

stringr package offers `str_()` functions to handle strings in R.

```r
library(stringr)
str_to_upper(string, locale = "")
str_to_lower(string, locale = "")
str_to_title(string, locale = "")
str_count(string, pattern = "")
str_length(string)
str_detect(string, pattern)
str_extract(string, pattern)
str_replace(string, pattern, replacement)
str_sub(string, start = 1L, end = -1L)
str_trim(string, side = c("both", "left", "right"))
```